Introduction machine learning on graphs

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실
NETWORK SCIENCE LAB

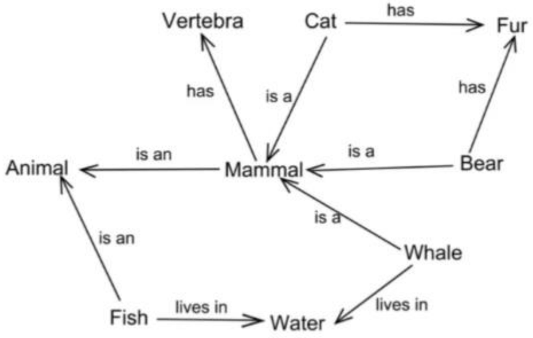
가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

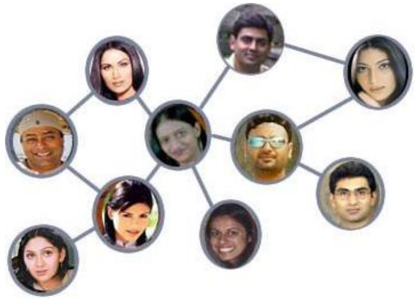
Graphs are everywhere

Networks are a general language for describing and modeling complex systems

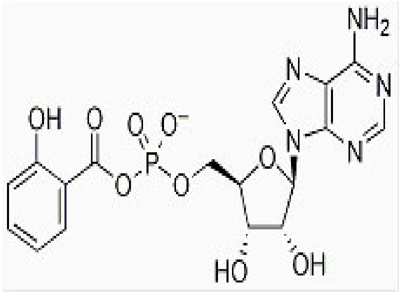
Street network



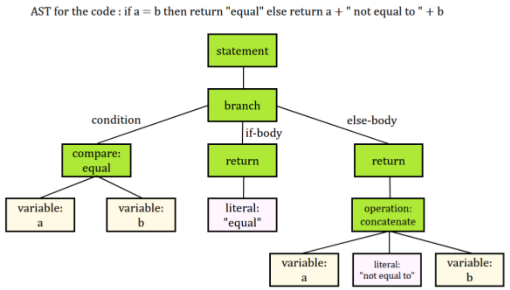
Ecological network



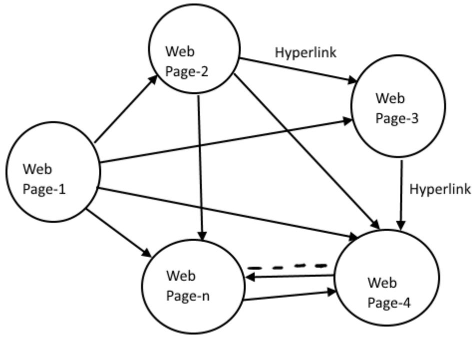
Social media



Chemical network



Program flow



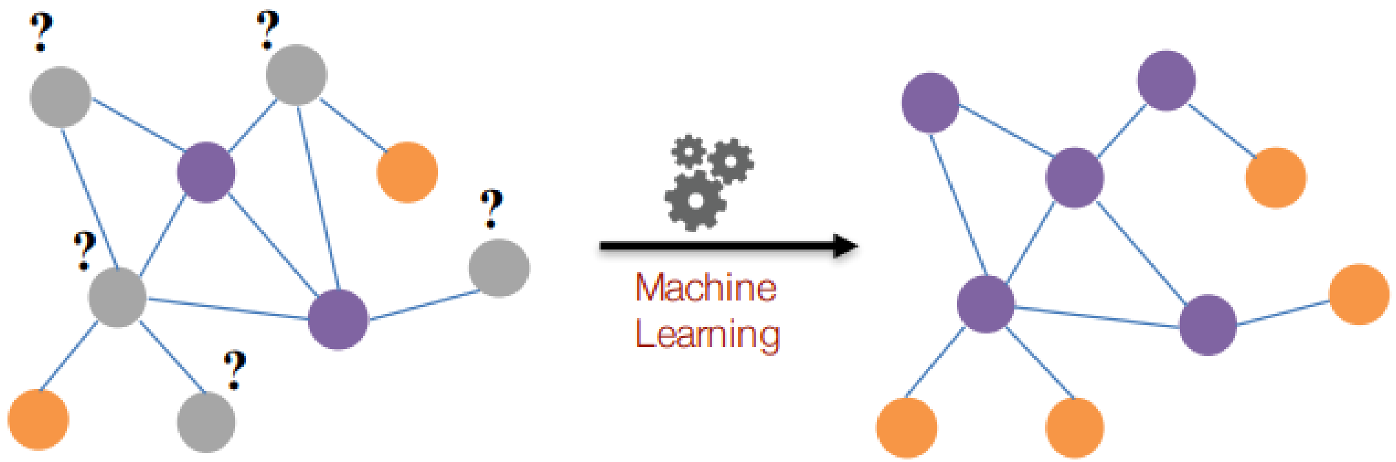
Web graph

Why Graph?

- Universal language for describing complex data
  - Chemical compounds (Cheminformatics)
  - Protein structures, biological pathways/networks (Bioinformactics)
  - Program control flow, traffic flow, and workflow analysis
- Data availability (+computational challenges)
  - Web/mobile, bio,health, and medical data
- Shared vocabulary between fields:
  - Computer science, Social science, Physics, Statistics, Biology
- Impact:
  - Social networking, social media, Drug design

Machine learning tasks on graphs

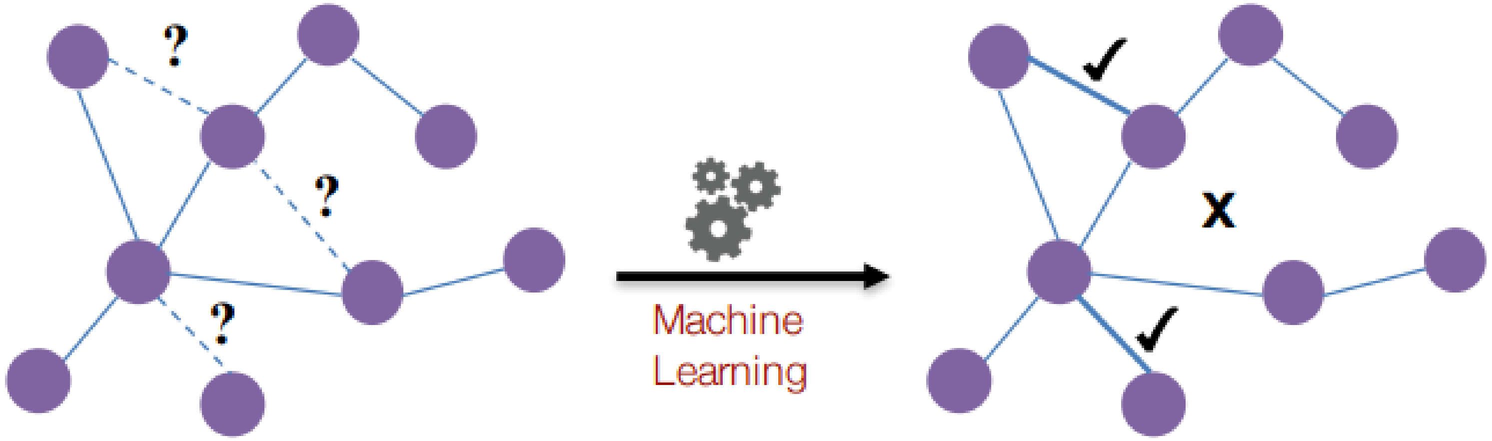
- Node classification
  - Predict a type of a given node



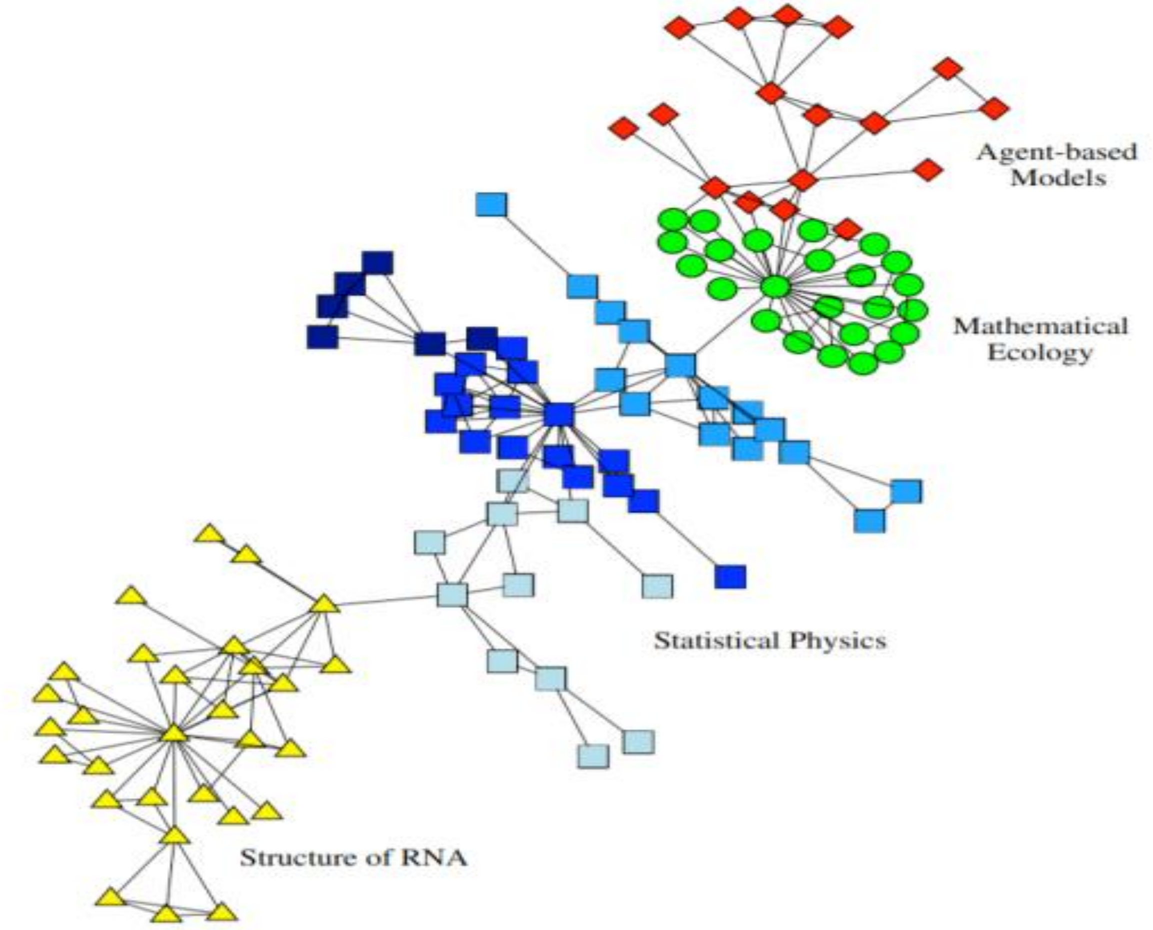
  - Many possible ways to create node features:
    - Node degree, PageRank score, motifs

Machine learning tasks on graphs

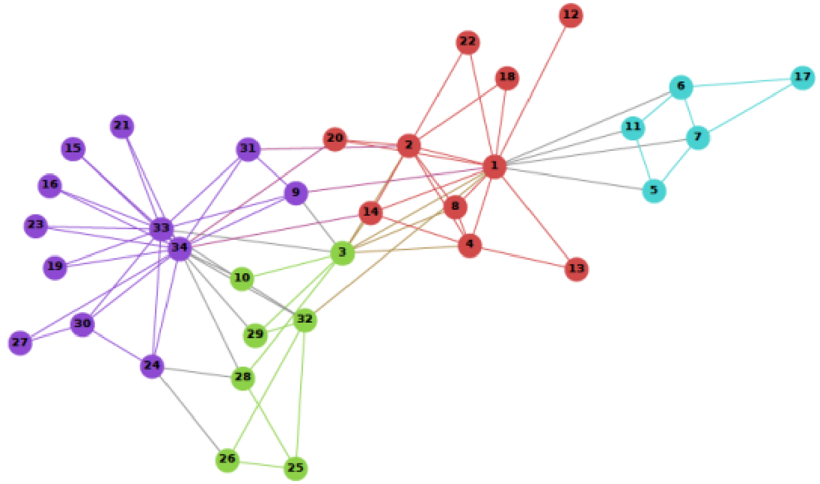
- Link prediction
  - Predict whether two nodes are linked

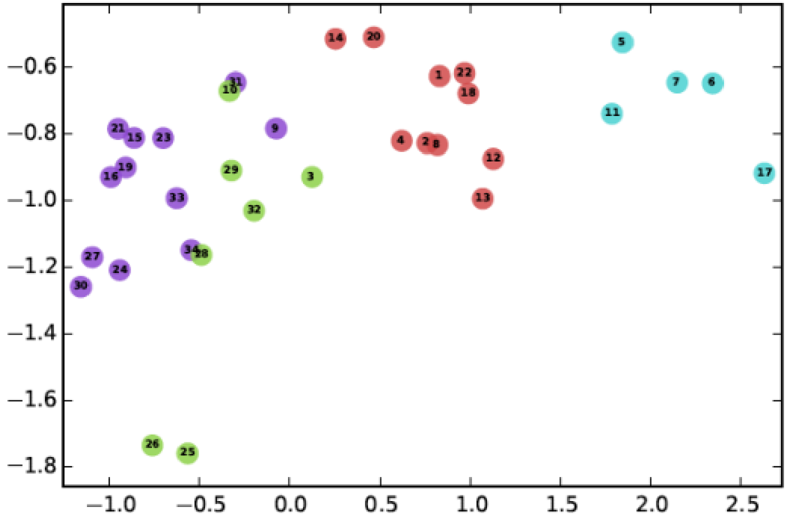


- Community detection
  - Identify densely linked clusters of nodes



➢ Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.





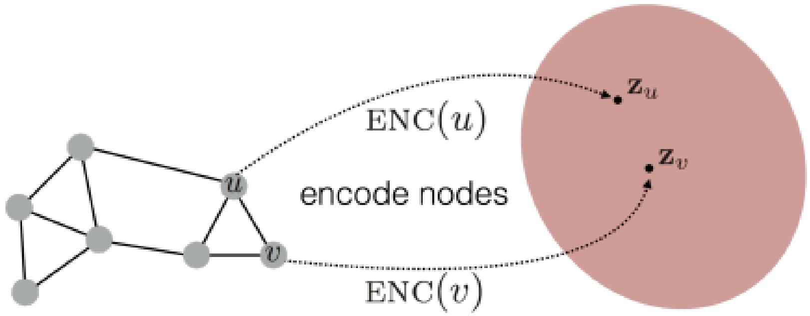
Node embedding

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.
- Let z_u be the embedding of node u.
- Goal is to find the encoder function f such that:

$$\text{similarity}(u, v) \approx z_u^T z_u$$

- Learning node embedding:
  - Define an encoder .
  - Define a node similarity function.
  - Optimize the parameters of the encoder so that:

similarity(u, v) ≈ $z_u^T z_u$

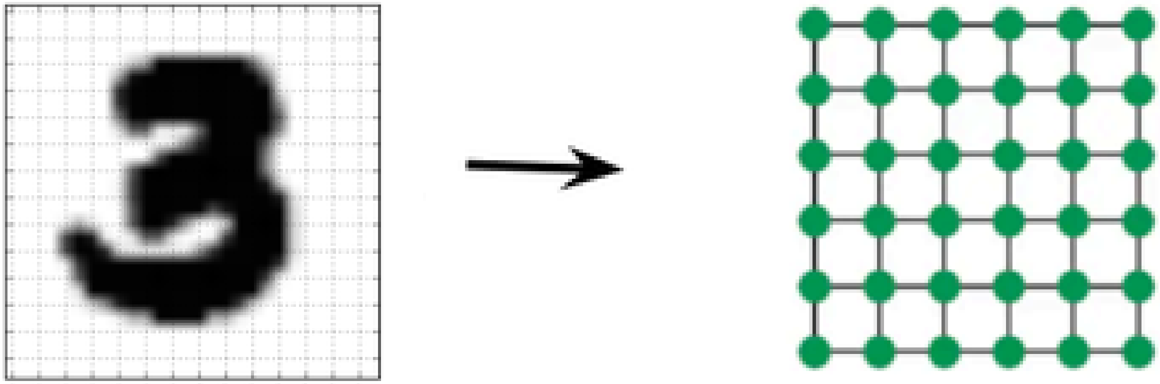


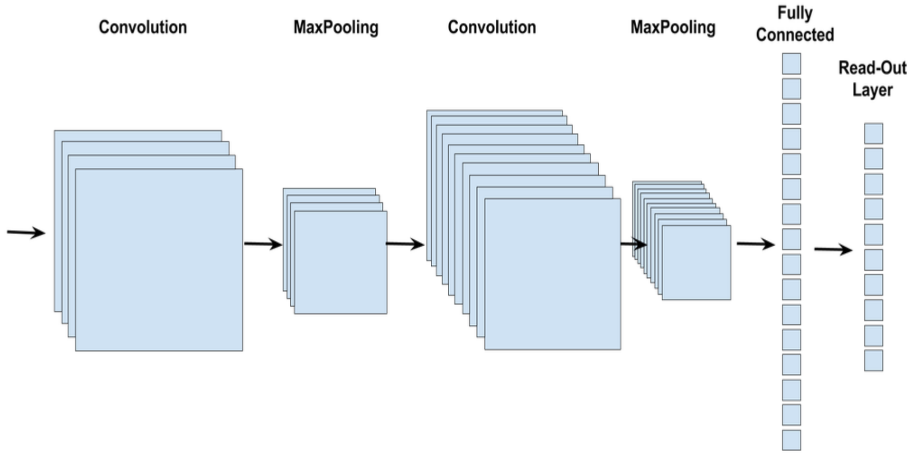
Why learn Embeddings?

- The goal is to map each node into a low-dimensional space
  - Distributed representation for nodes
  - Similarity between nodes indicates link strength
  - Encodes network information and generate node representation

Why is it hard to analyze a graph?

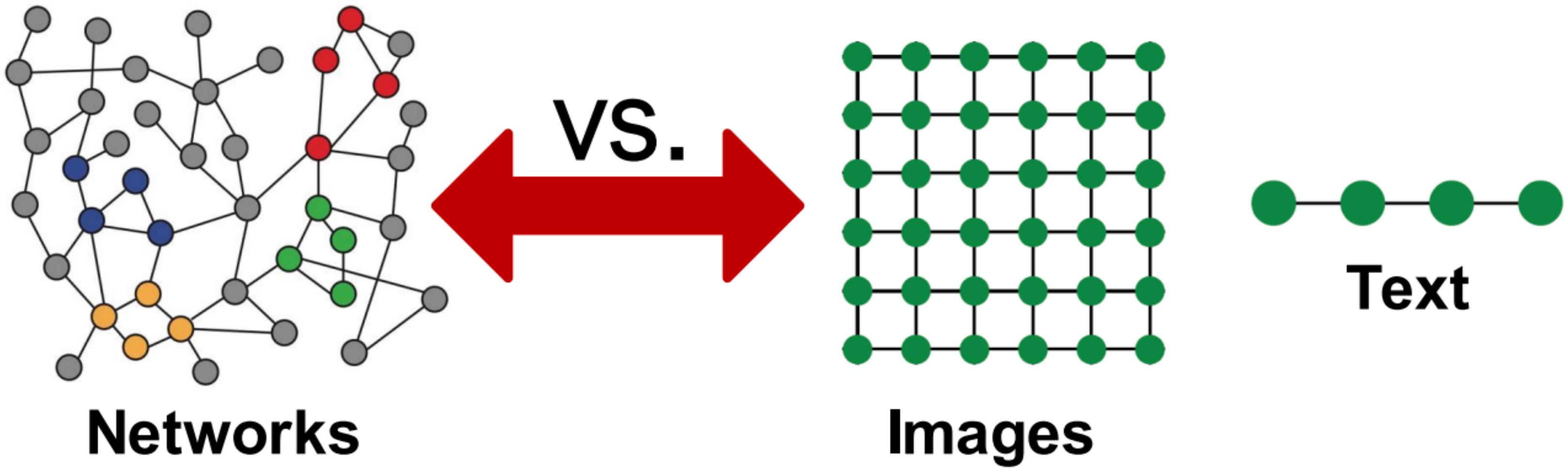
- Graph data is so complex that it's created a lot of challenges for existing machine learning algorithms.
- Images with the same structure and size can be considered as fixed-size grid graphs.
- Text and speech are sequences, so they can be considered as line graphs. (text and speech have linear 1D structure





Why is it hard to analyze a graph?

- Graphs have arbitrary size and complex topological structure.
- In graphs, there is no fixed node ordering or reference point.
- Graphs are often dynamic and have multimodal features.



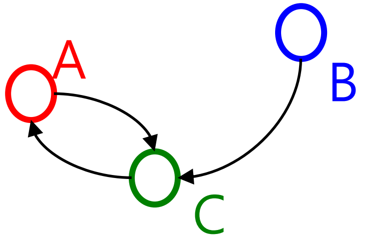
Our goal

- How can we develop neural networks that are much more broadly applicable?
- Feature learning for networks:
  - "Linearizing" the graphs:
    - Create a "sentence" for each node using random walks (node2vec)
  - Graph neural networks:
    - Propagate information between the nodes in graphs (message passing)

Graph Definition

A graph is a pair: G = (V, E):

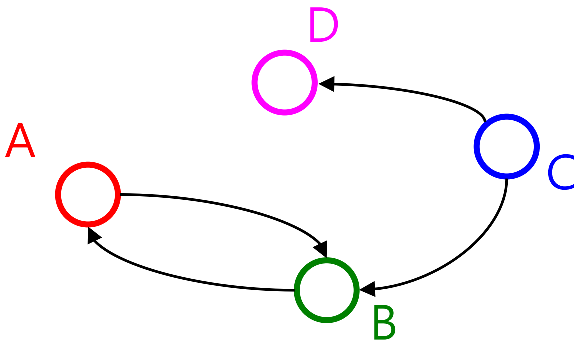
- ➢ A set of nodes, also known as nodes: $V = \{v_1, v_2, \ldots, v_n\}$
- ➢ A set of edges $E = \{e_1, e_2, \ldots, e_m\}$
  - ➢ Each edge e_i is a pair of nodes (v_j, v_k)
  - ➢ An edge "connects" the nodes



$V = \{ A, B, C \}$
$E = \{(B, C), (A, C), (C, A)\}$

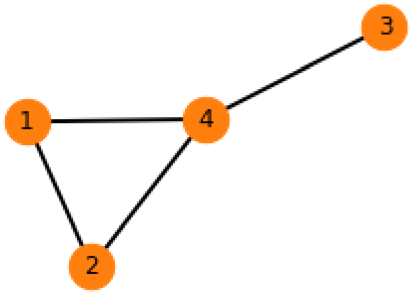
Graphs can be *directed* or *undirected*

- Adjacency Matrix

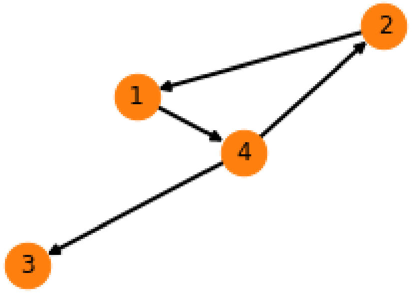


	A	B	C	D
A	0	1	0	0
B	1	0	0	0
C	0	1	0	1
D	0	0	0	0

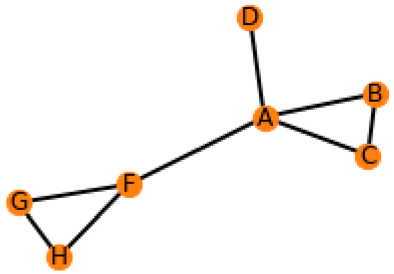
Graph Components and Types



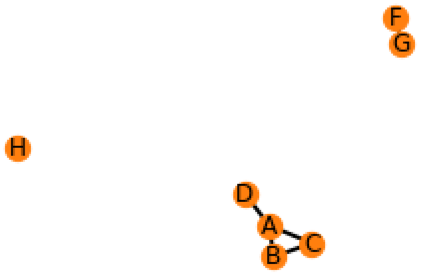
Undirected



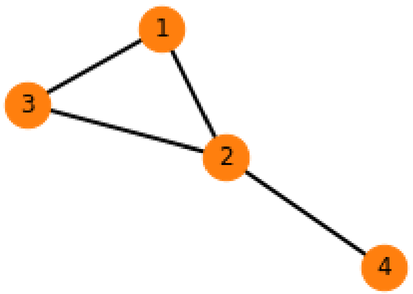
Directed



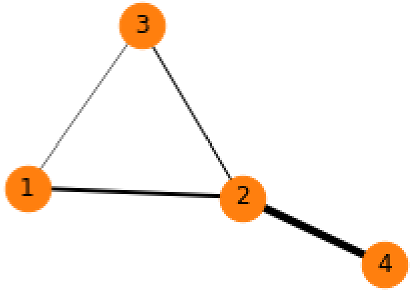
Connected



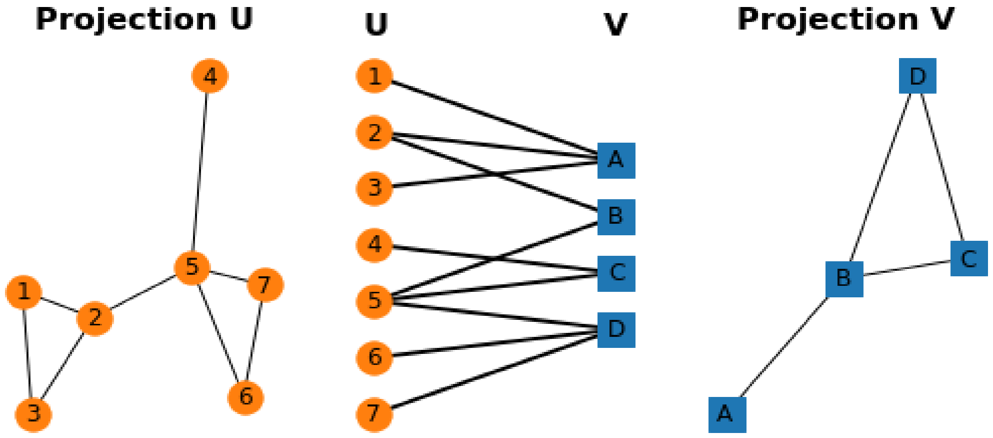
Disconnected



Unweighted

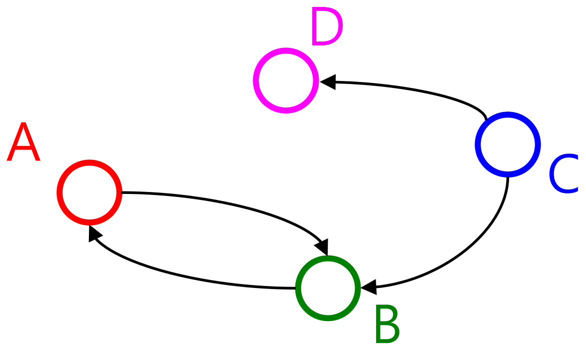


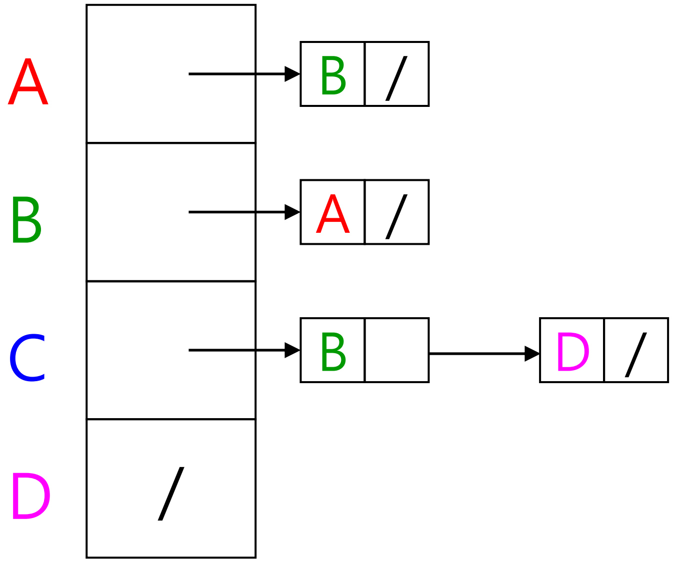
Weighted



Folded/Projected Bipartite Graphs

- Adjacency List





Adjacency List Properties

- Running time to:
  - Get a vertex's out-edges: O(*d*) where *d* is out-degree of vertex
  - Get a vertex's in-edges: O(|E|) (could keep a second adjacency list for this!)
  - Decide if some edge exists: O(*d*) where *d* is out-degree of source
  - Insert an edge: O(1) (unless you need to check if it's already there)
  - Delete an edge: O(*d*) where *d* is out-degree of source

- Space requirements: O(|V|+|E|)

- Best for sparse or dense graphs? sparse

Centrality Measures

- Knowing the network structure, we can calculate various useful quantities or measures that capture features of network topology
- Centrality measures represent the most important nodes in graphs:
  - The most influential person in a social network.
  - The most critical nodes in a infrastructure.
  - The highest spreaders of disease.
- Several common measurements:
  - Degree centrality
  - Betweenness centrality
  - Closeness centrality
  - Eigenvector centrality
  - PageRank

Degree centrality examples

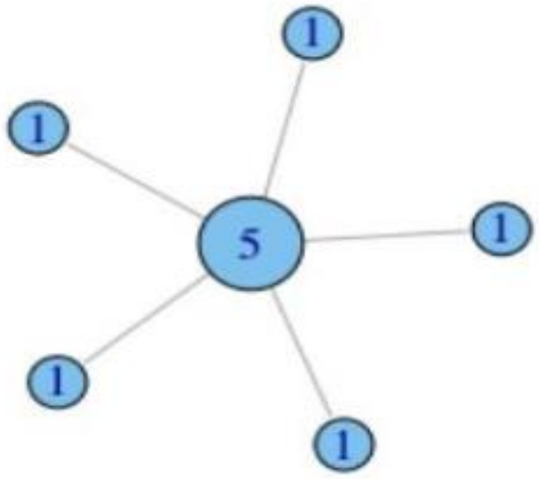
- Using Freeman's general formula for centralization (which ranges from 0 to 1):

$$C_D(G) = \frac{\sum_{i=1}^{n}\left[C_D(v^*) - C_D(v_i)\right]}{(\mathrm{n}-1)(\mathrm{n}-2)},$$

$$where:$$

$$v^* : \text{ the node with the highest degree in G}$$

$C_D = 0.167$



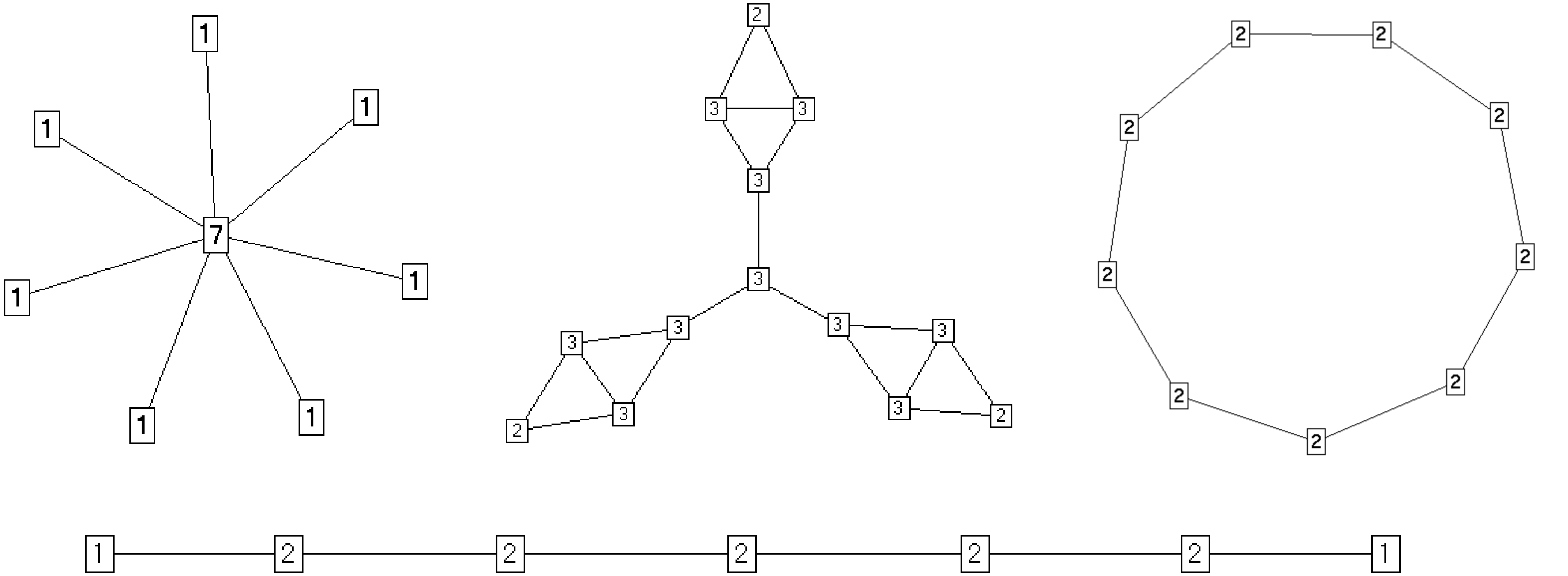
$C_D = 1.0$

$$C_D(G) = \frac{(2-1)+(2-0)+\ldots+(2-1)}{(5-1)(5-2)} = 0.167$$

$$C_D(G) = \frac{(5-1)+\ldots+(5-1)}{(6-1)(6-2)} = \frac{20}{20} = 1$$

Degree centrality

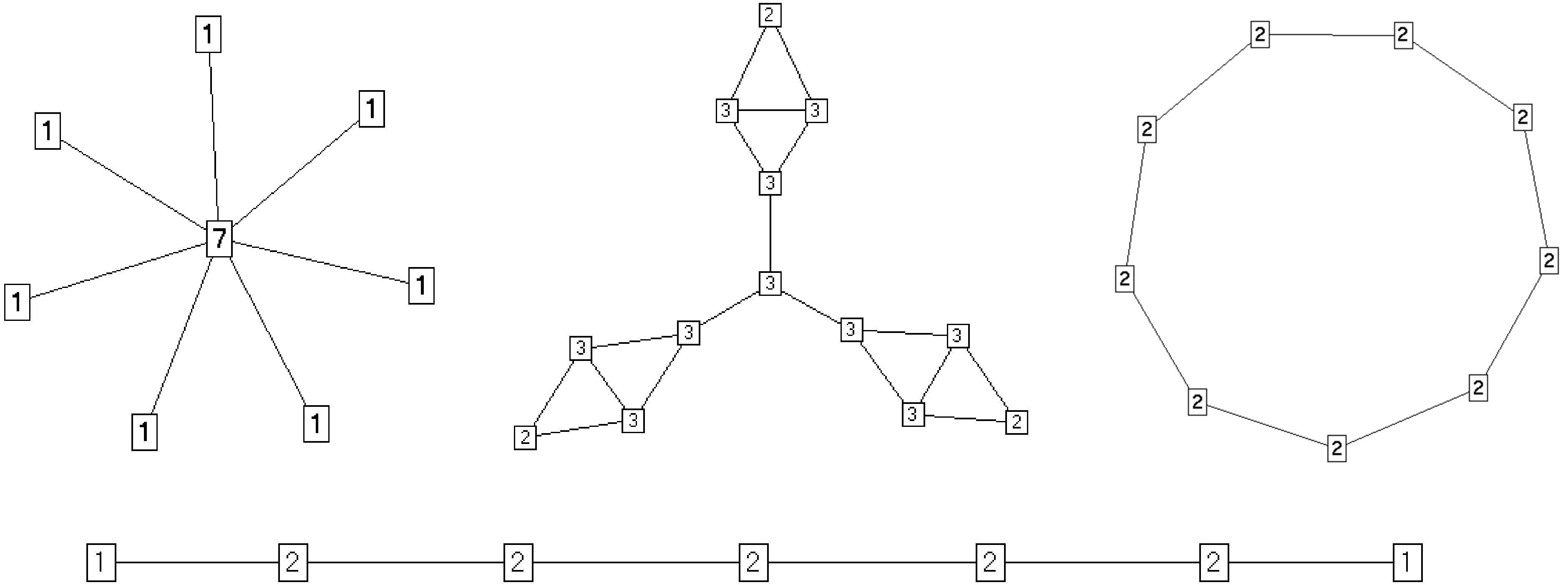
- ➢ The most intuitive notion of centrality focuses on degree:
  - ➢ The actor with the most ties is the most important:



$$C_D(v_i) = d(v_i) = \sum_{j=1}^{n} A_{ij}$$

Degree centrality

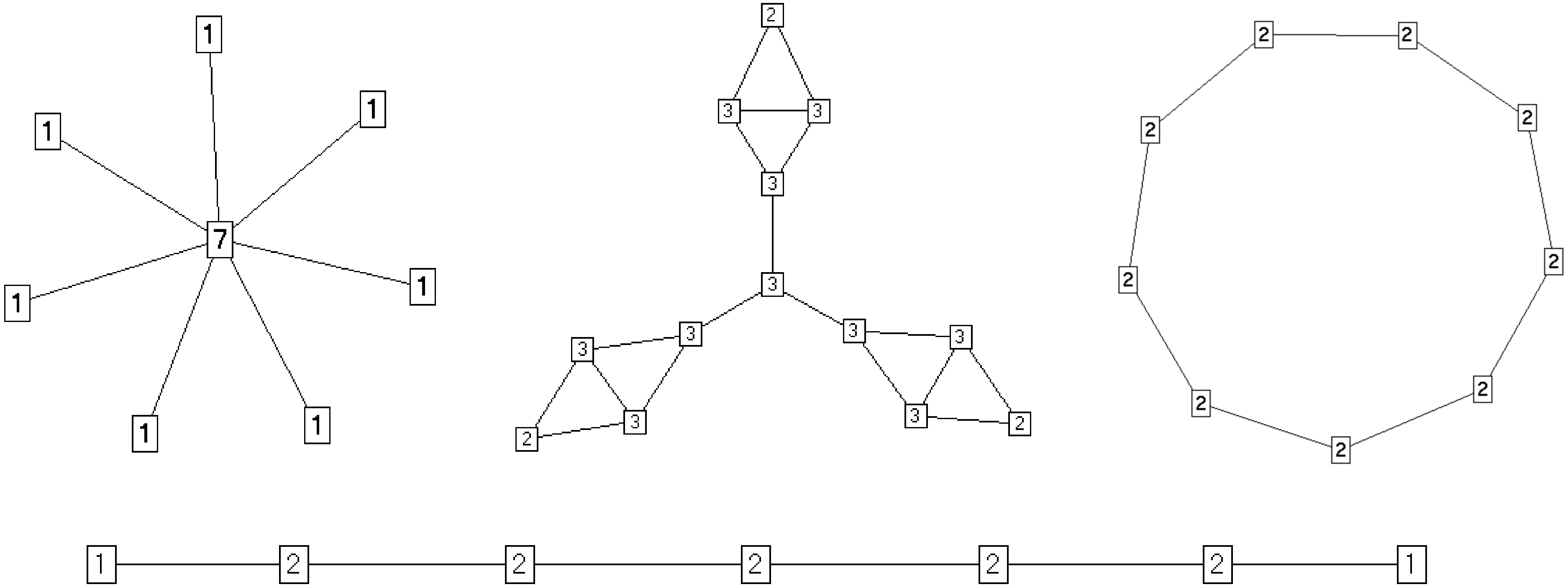
- The most intuitive notion of centrality focuses on degree:
  - The actor with the most ties is the most important:



$$C_D(v_i) = d(v_i) = \sum_{j=1}^{n} A_{ij}$$

Degree centrality

- The most intuitive notion of centrality focuses on degree:
  - The actor with the most ties is the most important:



$$C_D(v_i) = d(v_i) = \sum_{j=1}^{n} A_{ij}$$

Betweenness centrality

- Betweenness centrality of node v_i :

The number of shortest paths between v_j and v_k that pass through the vertex v_i

$$B(v_i) = \sum_{v_j, v_k \in G} \left| SPD_{v_j \to v_k}(v_i) \right|$$

- Usually normalized by:

No. pairs of nodes excluding the node itself

$$\bar{B}(v_i) = B(v_i) / [(n-1)(n-2)/2]$$

Closeness centrality

- The closeness is defined so that if a vertex is close to every other vertex, then the value is larger than if the vertex is not close to everything else.

SPD: The number of nodes in the shortest path between 2 nodes

$$C(v_i) = \left[\sum_{j=1}^{n} \left| \mathrm{SPD}(v_i, v_j) \right| \right]^{-1}$$

- Normalized Closeness Centrality:

$$\bar{C}(v_i) = C(v_i) / (\mathrm{n}-1)$$

- Define the centrality x'_i of i recursively in terms of the centrality of its neighbors:

$$x'_i = \sum_{v_j \in N(v_i)} A_{ij} x_j \qquad \text{with the initial node centrality } x_j = 1, \forall j$$

- That is equivalent to:

$$x_i(\mathrm{t}) = \sum_{v_j \in N(v_i)} A_{ij} x_j(\mathrm{t}-1) \qquad \text{with the centrality at time t=0 being } x_j(0) = 1, \forall j$$

The centrality of nodes x_i and x_j at time *t* and *(t-1)*, respectively.

Katz centrality

- Katz centrality computes the centrality for a node based on the centrality of its neighbours. It is a generalization of the eigenvector centrality.
- The Katz centrality for node v_i is:

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where:

α is a constant called damping factor, and β is a bias constant,

A is the adjacency matrix.

- When $\alpha = 1/\lambda_{\max}, \beta = 0$, Katz centrality is the same as eigenvector centrality

PageRank

PageRank is a numeric value that represents how important a page is on the web.

- Webpage importance
  - One page links to another page = A vote for the other page A link from page A to page B is a vote on A to B.
  - If page A is more important itself, then the vote of A to B should carry more weight.
  - More votes = More important the page must be

- How can we model this importance?

Taxonomy of Community Criteria

- Criteria vary depending on the tasks.
- Roughly, community detection methods can be divided into 4 categories (not exclusive):

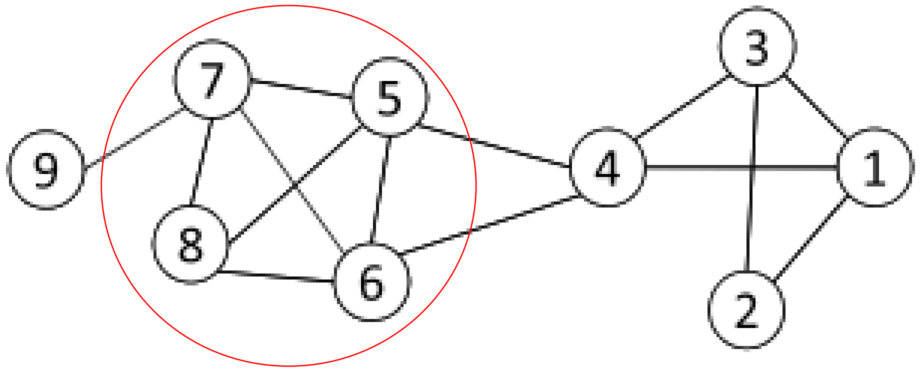
- 1. Node-Centric Community
  - Each node in a group satisfies certain properties
- 2. Group-Centric Community
  - Consider the connections within a group as a whole. The group has to satisfy certain properties without zooming into node-level
- 3. Network-Centric Community
  - Partition the whole network into several disjoint sets
- 4. Hierarchy-Centric Community
  - Construct a hierarchical structure of communities

- Nodes satisfy different properties
  - Complete Mutuality
    - cliques
  - Reachability of members
    - k-clique, k-clan, k-club
  - Nodal degrees
    - k-plex, k-core
  - Relative frequency of Within-Outside Ties
    - LS sets, Lambda sets
- Commonly used in traditional social network analysis

We discuss some representative ones

Complete Mutuality: Cliques

- Clique: a maximum complete subgraph in which all nodes are adjacent to each other



Nodes 5, 6, 7 and 8 form a clique

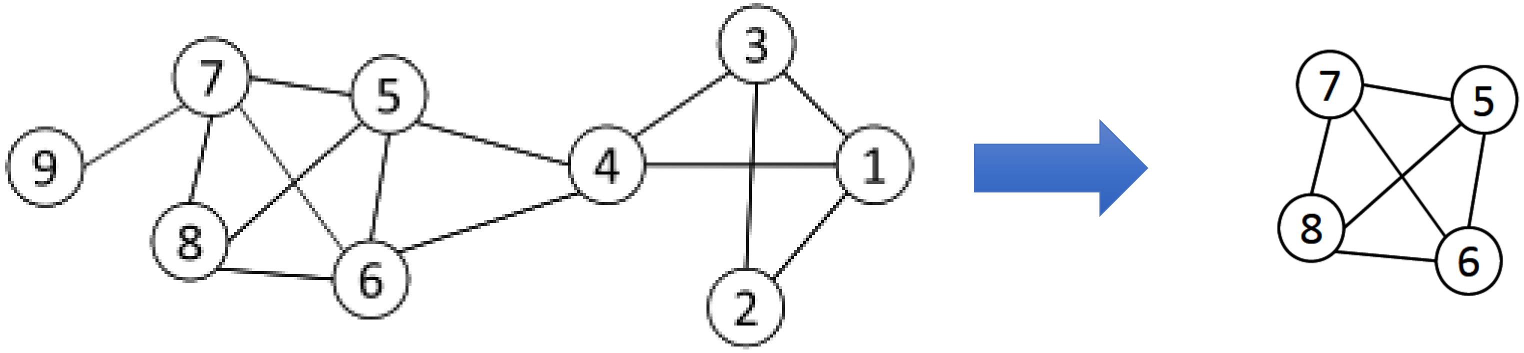
- NP-hard to find the maximum clique in a network
- Straightforward implementation to find cliques is very expensive in time complexity

Finding the Maximum Clique

- In a clique of size k, each node maintains degree >= k-1
- Nodes with degree < k-1 will not be included in the maximum clique
- Recursively apply the following pruning procedure:
    - Sample a sub-network from the given network, and find a clique in the sub-network, say, by a greedy approach
    - Suppose the clique above is size k, in order to find out a larger clique, all nodes with degree <= k-1 should be removed.
- Repeat until the network is small enough
- Many nodes will be pruned as social media networks follow a power law distribution for node degrees

Maximum Clique Example

- Suppose we sample a sub-network with nodes {1-5} and find a clique {1, 2, 3} of size 3
- In order to find a clique >3, remove all nodes with degree <= 3 – 1 = 2
  - Remove nodes 2 and 9
  - Remove nodes 1 and 3
  - Remove node 4

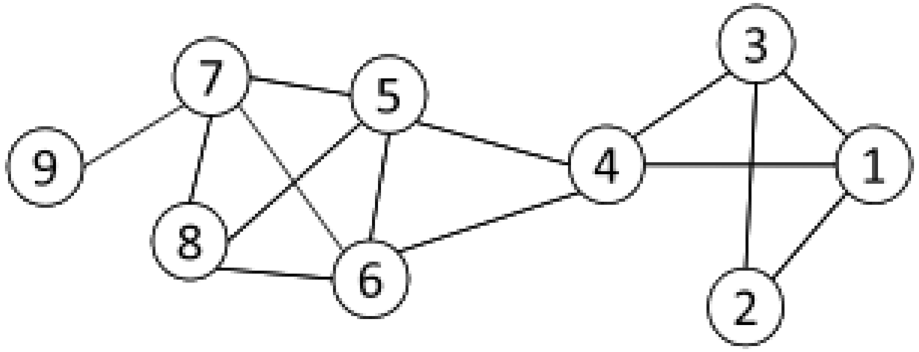


Clique Percolation Method (CPM)

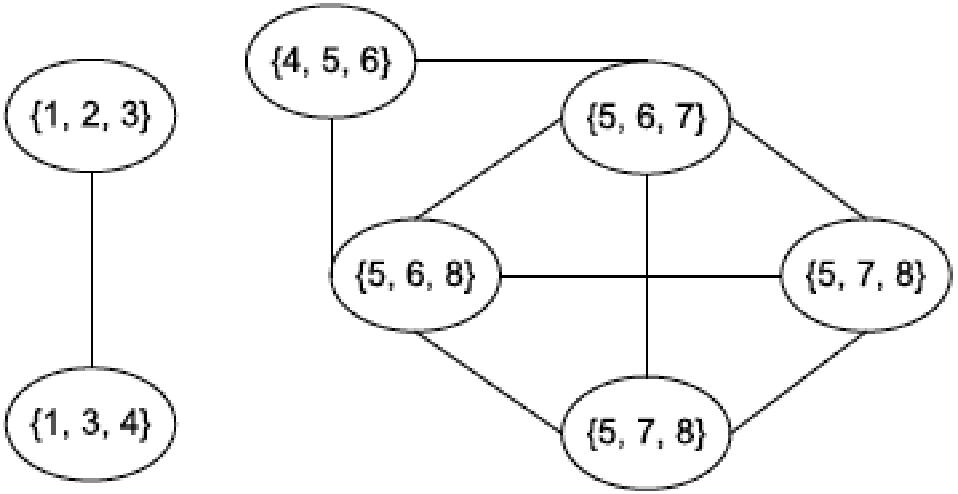
- Clique is a very strict definition, unstable
- Normally use cliques as a core or a seed to find larger communities

- CPM is such a method to find overlapping communities
- Input
  - A parameter k, and a network
- Procedure
  - Find out all cliques of size k in a given network
  - Construct a clique graph. Two cliques are adjacent if they share k-1 nodes
  - Each connected components in the clique graph form a community

Clique Percolation Method (CPM): Example



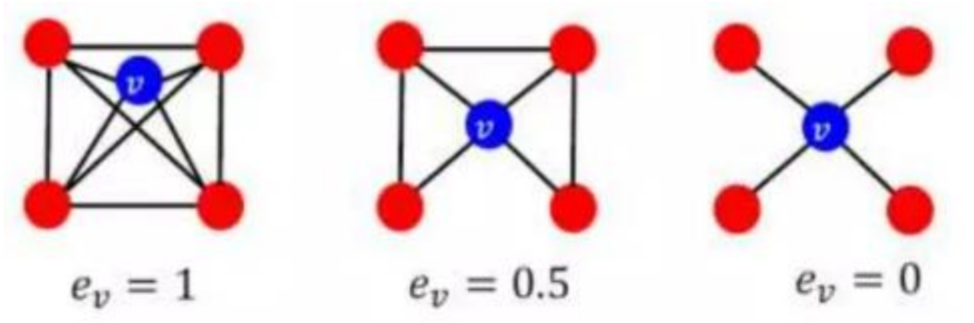
Cliques of size 3:
{1, 2, 3}, {1, 3, 4}, {4, 5, 6},
{5, 6, 7}, {5, 6, 8}, {5, 7, 8},
{6, 7, 8}



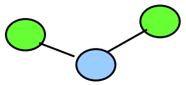
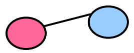
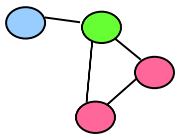
Communities:
{1, 2, 3, 4}
{4, 5, 6, 7, 8}

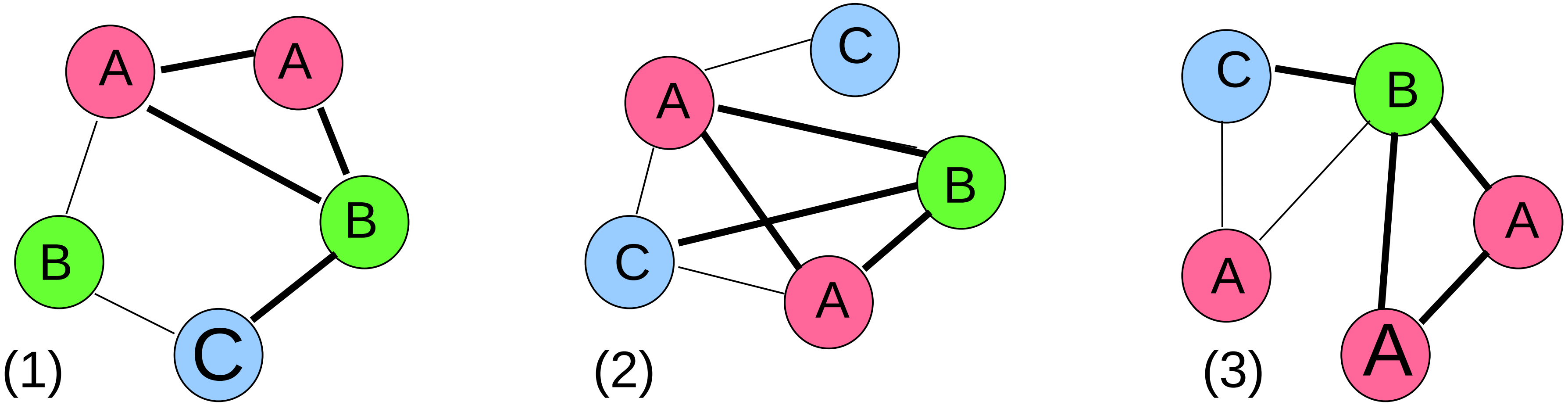
Node Features

- Degree of nodes
- Clustering Coefficient
  - Measures how connected neighboring nodes are
  - E.g., The number of edges among neighboring nodes

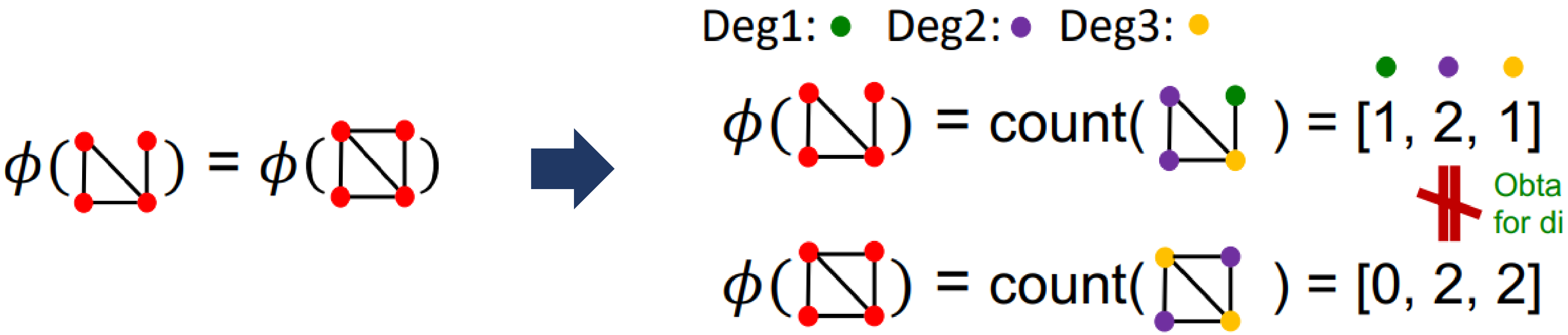


- Frequent subgraphs
  - A (sub)graph is frequent if its support (occurrence frequency) in a given dataset is no less than a minimum support threshold
  - Suppose t = 2, the frequent subgraphs are (only edge labels)
    - a, b, c
    - a-a, a-c, b-c, c-c
    - a-c-a …

Support	1	3	3
Subgraph			

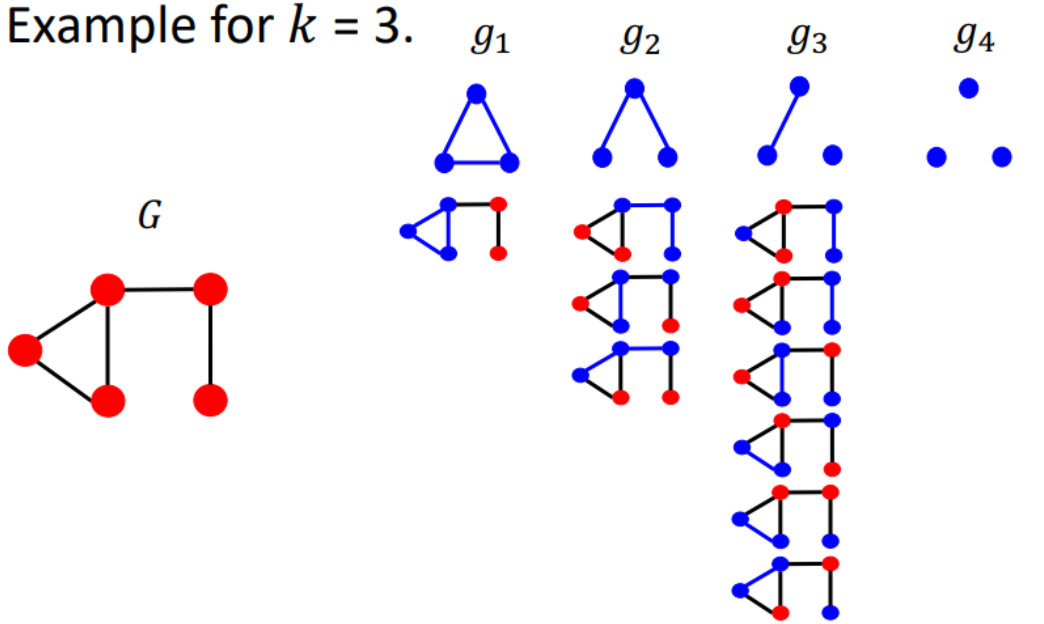


- Graph kernels based on bags of patterns:
  - Extraction of a set of patterns from graphs
  - Comparison between patterns
  - Comparison between bags of patterns



- Graphlet Kernel (B., Petri, et al., MLG 2007)
- Count subgraphs of limited size 3:

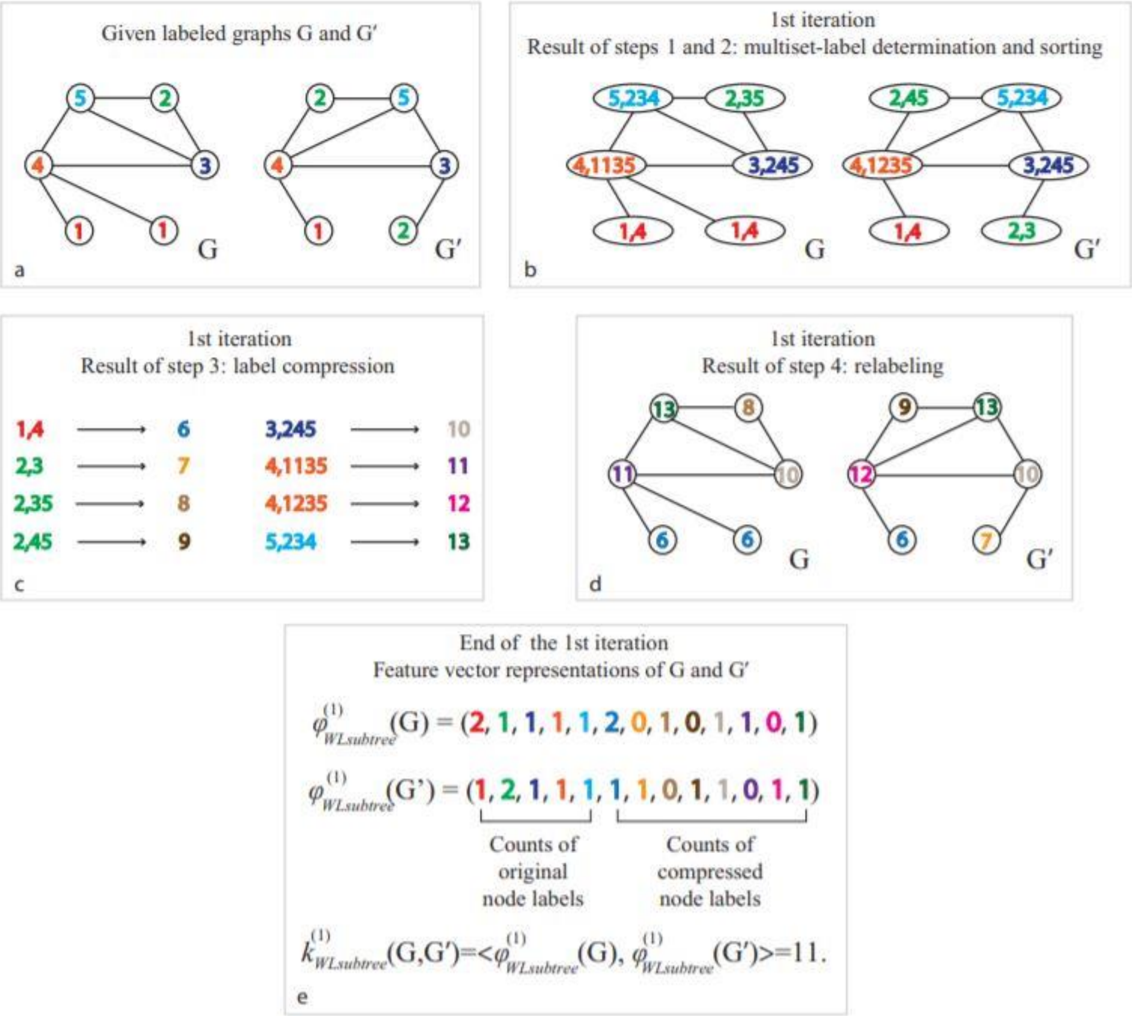
- Example for k = 3.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^T$$

The Weisfeiler-Lehman Isomorphism Test

- Weisfeiler-Lehman Isomorphism Testing:

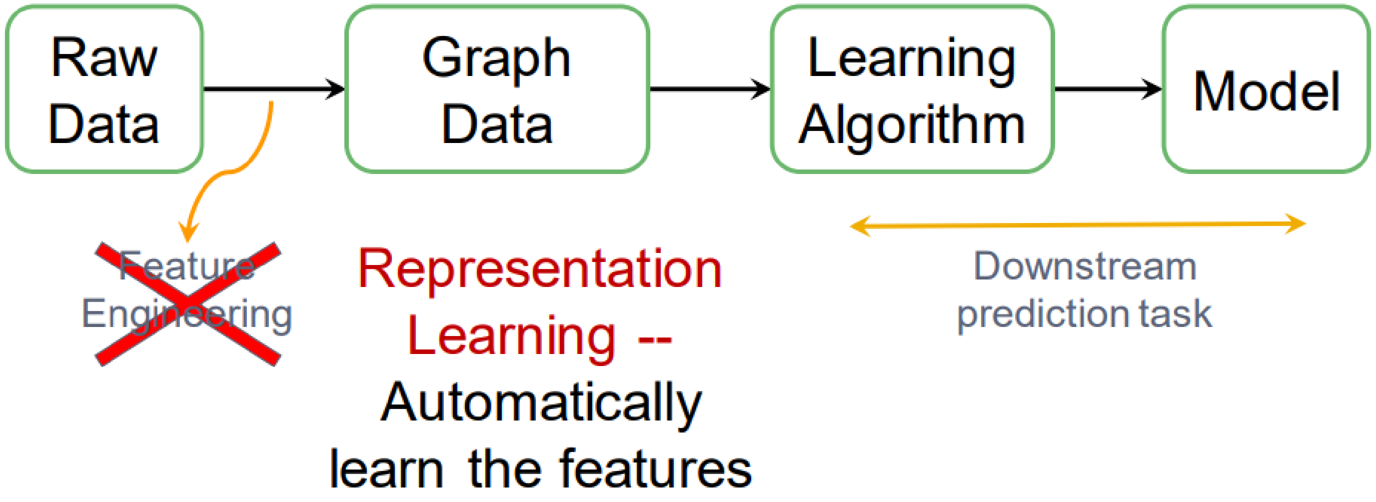


Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
t $\leftarrow 0$;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
until *stable node coloring is reached*;

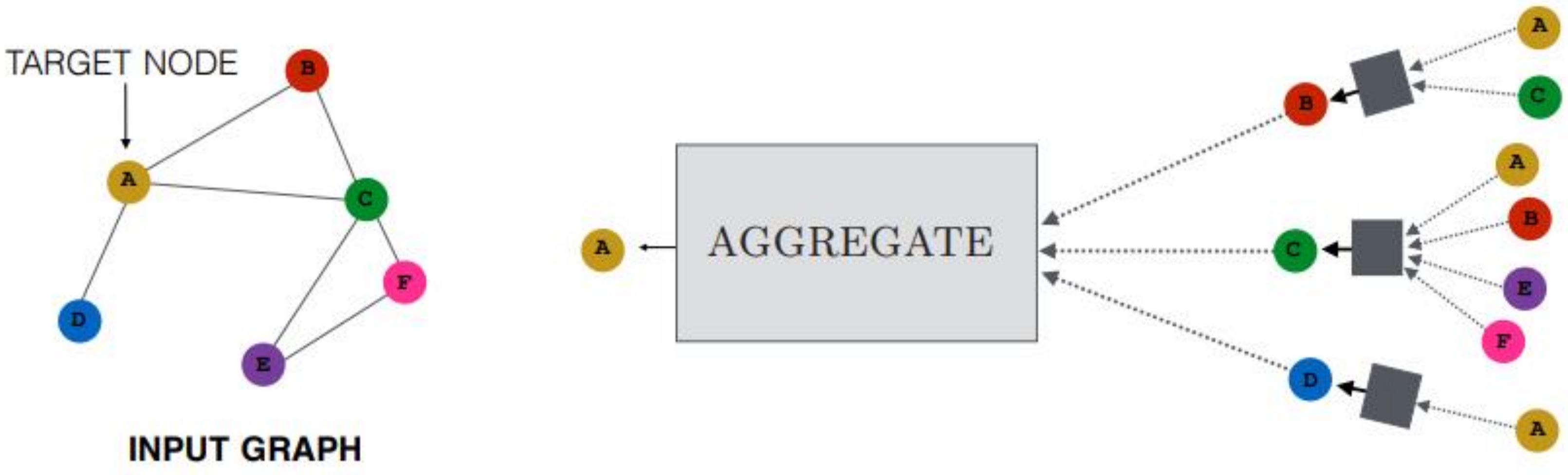
Representation Learning

- (Supervised) Machine Learning Lifecycle
- This feature, that feature. Every single time!

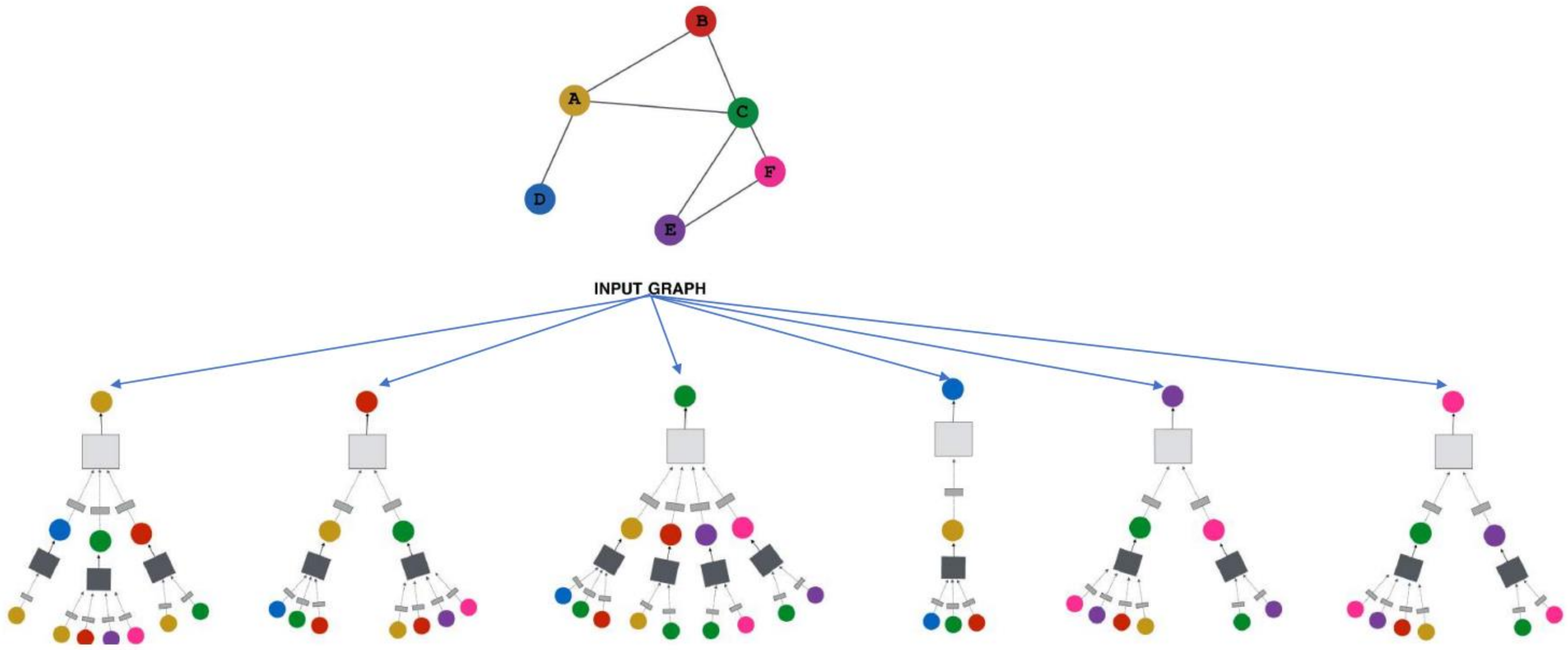


Graph neural networks (GNNs)

- The idea is to generate node embeddings based on local neighborhoods
- The intuition is nodes aggregate information from their neighbors using neural networks.



➢ Network neighborhood defines a computation graph



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA